

FIGURE 1

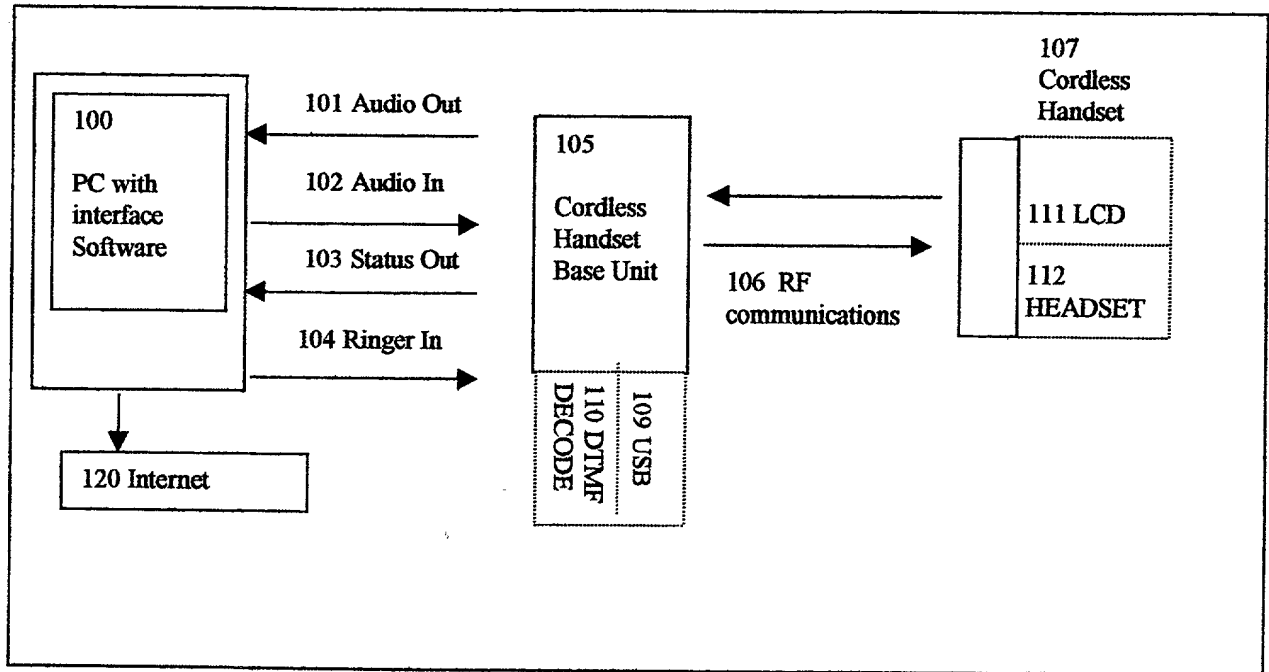
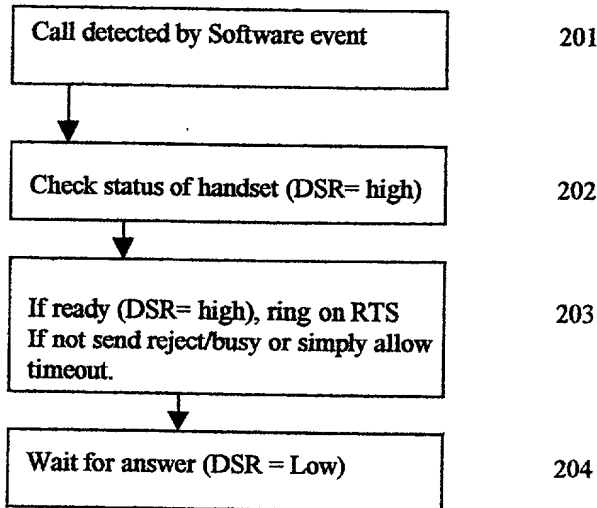
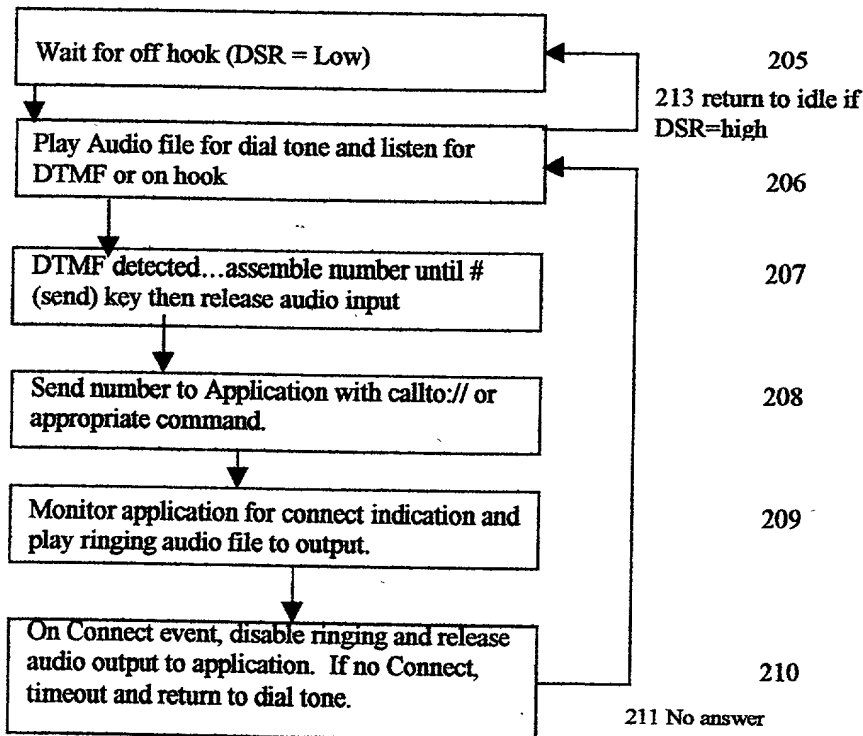


FIGURE 2

Received call



Outgoing Call



200 Initial Setup Steps:

1. Connect phone on a COM port
2. Connect Audio cables
3. Select default sound system mixer (if more than one.
4. Set volumes to 50% for Record Mic and Playback Wav , disable Playback Mic and Record Wav to prevent feedback.
5. Run Internet Telephony software to allow it to get control of the sound system
6. Run sample control program (fig 5)
7. Verify that program detects phone on com port and test ring it.
8. Choose Internet Telephony Service
9. Enable Dialer
10. Hit Talk button to play dialtone and trigger auto recovery if sound system is set up incorrectly (no default device).

Figure 3

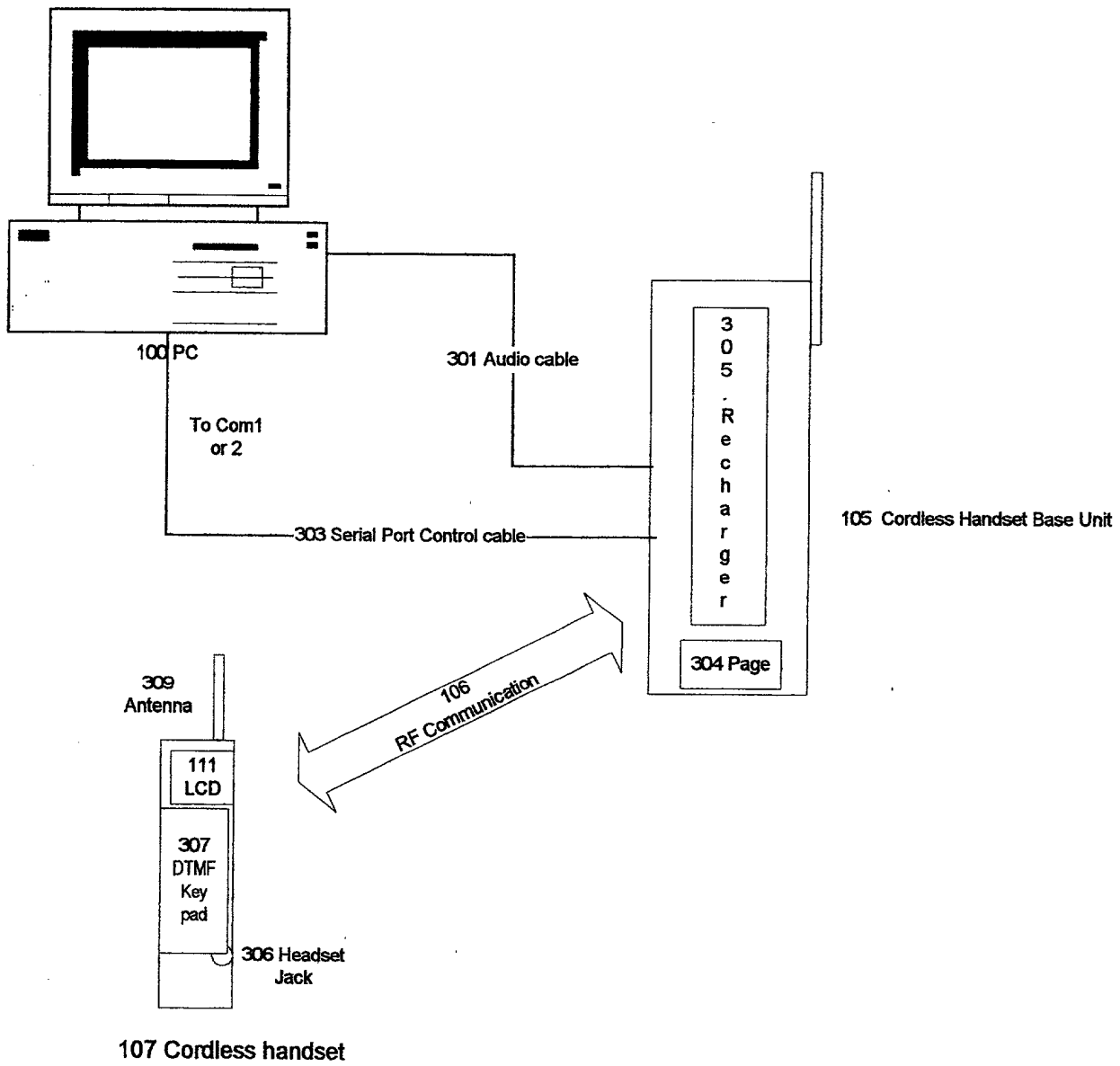


Figure 4 Simplified Block Diagram

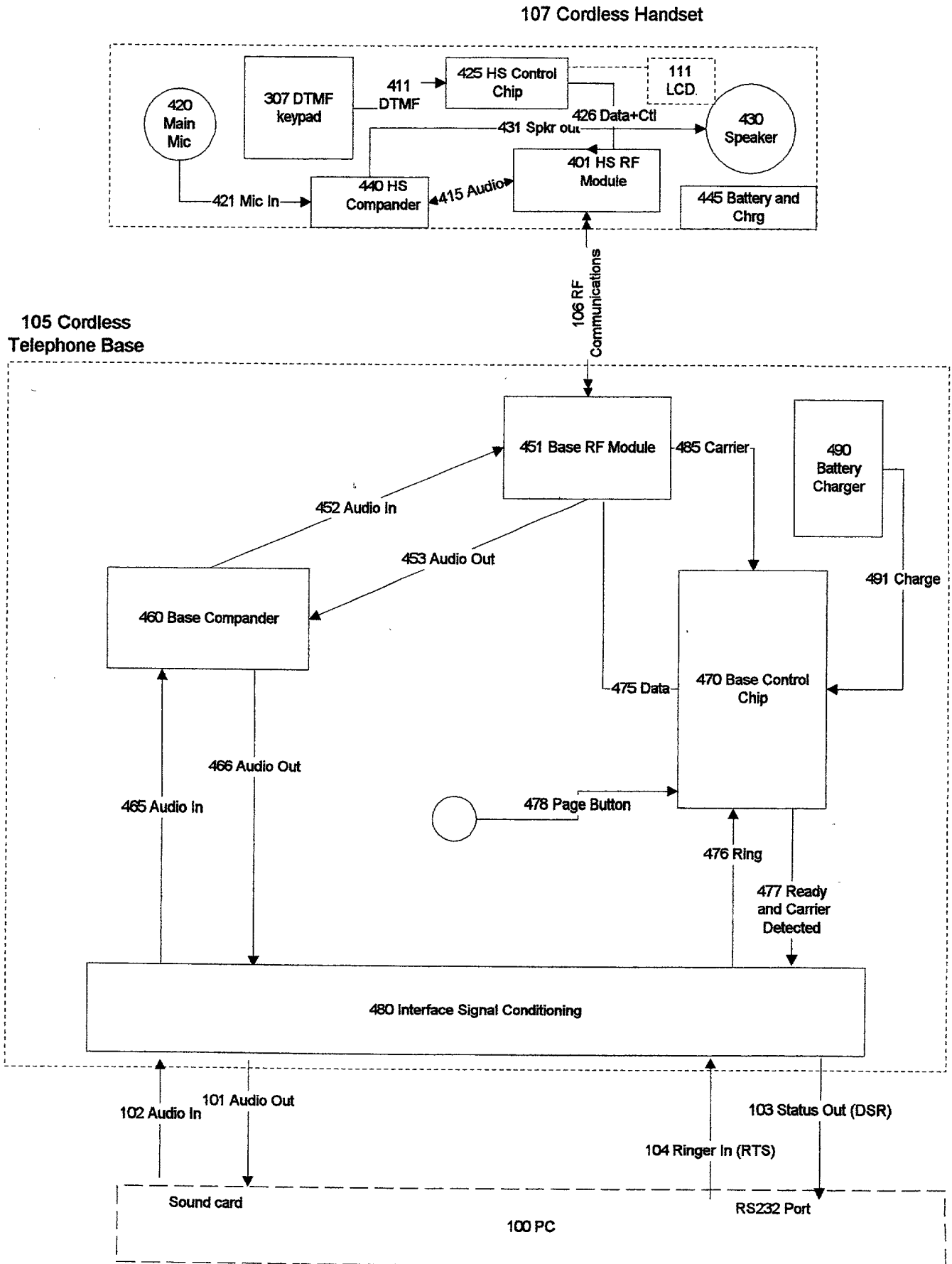


Figure 5 Graphical user interface

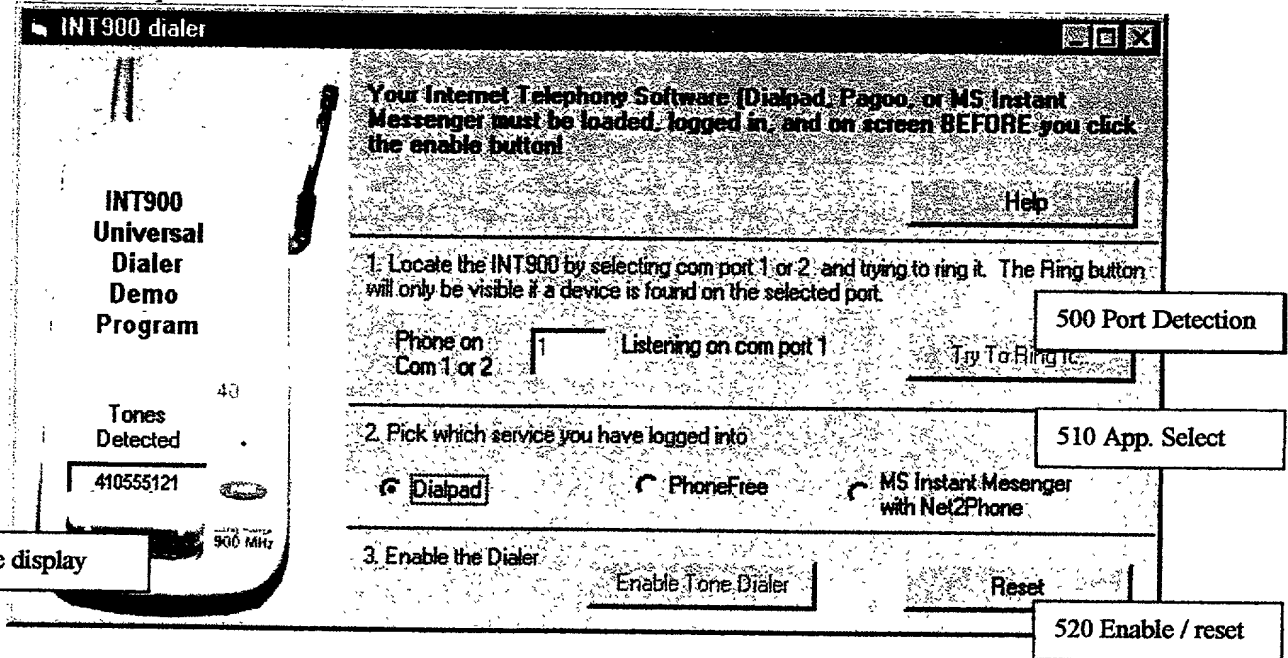


Figure 6 Visual Basic Program Listing

'Sample listing for setup of DTMF recognition and serial port operations related to use of the Cordless Internet Telephone to emulate a POTS call using one of several service providers.

Misc. (Types, Enums, Constants...)

' Means to use a default or automatic value
Global Const ttAutomatic = -1

' Errors used by ttLib

Enum ttErrors

tteNoError = 0

tteUnknown

tteNotSupported

tteOutOfMemory

tteWaveDeviceDidNotOpen

tteWrongState

tteBadParameter

End Enum

' Most ttLib functions return a success value

Enum ttSuccessValue

ttscFailure = 0

ttscSuccess = -1

End Enum

' Used by ttInitializeExInfo and ttInitialize()

Enum ttCallbackMethods

ttcmPostMessage = 0 ' PostMessage to a given window

ttcmSendMessage = 1 ' SendMessage to a given window

ttcmDirectFunction = 2 ' Call a function DIRECTLY FROM THE RECOGNIZER THREAD

End Enum

' Used by ttInitializeEx()

Type ttInitializeExInfo

StructSize As Long ' LenB(ttInitializeExInfo)

CallbackMethod As ttCallbackMethods ' Desired form of callback

CallbackInfo1 As Long ' If CB method is DirectFunction, this is a

Function pointer of a CallbackType function.

' If CB method is Send/PostMessage, this is

the target hWnd.

CallbackInfo2 As Long ' If CB method is Send/PostMessage, this is the

window message to send or ttAutomatic.

MinNoise As Long ' Noise gate level or ttAutomatic

ToneDuration As Long ' Number of 12.75ms sampling windows needed

for tone or ttAutomatic

SilenceDuration As Long ' Number of 12.75ms sampling windows needed

for silence or ttAutomatic

SignalToNoise As Long ' Signal to noise ratio or ttAutomatic

RowRatio As Long ' Primary to harmonic ratio for rows or

```

        ' ttAutomatic
ColRatio As Long        ' Primary to harmonic ratio for columns or
        ' ttAutomatic
DeviceOverride As Long  ' Audio input device to use or ttAutomatic
End Type

```

```

' Interface functions

```

```

' You call Initialize() or InitializeEx() when you first load the DLL or
' when you want to change some of the parameters. They are NOT meant to
' be called while recognition is in progress. The results of calling them
' while recognition is in progress are undefined.
' The ttInitializeExInfo struct passed to InitializeEx() will be returned
' with all ttAUTOMATIC values replaced by their defaults. The struct can
' freed immediately.

```

```

Declare Function ttInitialize Lib "ttLib" (ByVal Method As ttCallbackMethods, ByVal CallbackInfo1 As
Long) As ttSuccessValue

```

```

Declare Function ttInitializeEx Lib "ttLib" (ttLibInitializeExInfo As ttInitializeExInfo) As ttSuccessValue

```

```

' Starts up recognition

```

```

Declare Function ttBeginRecognition Lib "ttLib" () As ttSuccessValue

```

```

' Stops recognition. Failing to call this is an easy way to crash your app.

```

```

Declare Function ttEndRecognition Lib "ttLib" () As ttSuccessValue

```

```

' Returns an error value.

```

```

Declare Function ttGetErrorVal Lib "ttLib" () As ttErrors

```

```

' This is the prototype to use for callbacks functions. With VB6, this is
' useless unless you really know what you're doing. With VB5, it should
' be a breeze.

```

```

' Sub CallbackProc(ByVal Char As Byte)

```

The following Communications constants are from win32api.txt:

```

'EscapeCommFunction values

```

```

Global Const SETXOFF = 1

```

```

Global Const SETXON = 2

```

```

Global Const SETRTS = 3

```

```

Global Const CLRRTS = 4

```

```

Global Const SETDTR = 5

```

```

Global Const CLRDTR = 6

```

```

Global Const RESETDEV = 7

```

```

Global Const GETMAXLPT = 8

```

```

Global Const GETMAXCOM = 9

```

```

Global Const GETBASEIRQ = 10

```

'Global variables & constants

Public CommPorts() As String
Public PortExists As Boolean
Public Com1PortInUse As Boolean
Public Com2PortInUse As Boolean
Public PortNumber As Integer
Public PortOpen As Boolean
Public NumberOfPorts As Integer

Public ValidPort As Boolean
Public dtmf As String
Public getting_tone As Boolean

'API declares:

Declare Function PlaySound _
Lib "winmm.dll" _
Alias "PlaySoundA" _
(ByVal lpszName As String, ByVal hModule As Long, ByVal dwFlags As Long) As Long

Public Declare Function EscapeCommFunction _
Lib "kernel32" _
(ByVal nCid As Long, _
ByVal nFunc As Long) _
As Long

Public Declare Function timeGetTime _
Lib "winmm.dll" () _
As Long

Public Sub Delay(DelayInMilliseconds As Single)
'Delay timer with approximately 1-msec. resolution.
'Uses the API function timeGetTime.
'Rolls over 24 days after the last Windows startup.
Dim Timeout As Single
Timeout = DelayInMilliseconds + timeGetTime()
Do Until timeGetTime() >= Timeout
DoEvents
Loop
End Sub

Public Sub ShutDown()
'Close the port.
If Form1.MSComm1.PortOpen = True Then
Form1.MSComm1.PortOpen = False

End If
End Sub

Private Const WM_LBUTTONDOWN = &H202
Public dialing As Boolean
Public dialtoneplaying As Boolean
Public talking As Boolean

Private Sub CallbackTarget_MouseUp(Button As Integer, Shift As Integer, X As Single, Y As Single)
' When the recognizer gets a valid tone, it sends a window message with
' both the LPARAM and the WPARAM as the ASCII value of the key. Since we
' specified WM_LBUTTONDOWN as the message and CallbackTarget.hWnd as the window
' in our call to ttInitializeEx() -- this MouseUp event ends up firing when
' a key is recognized!
' For a normal WM_LBUTTONDOWN message, the X position of the mouse is in the
' low word of the LPARAM. Since VB uses that for the X parameter... the
' key's ASCII value is now in X! Note that CallbackTarget's ScaleMode is
' vbPixels. If it was something else, VB would have multiplied the real
' low word of LPARAM by something to make X (i.e., if ScaleMode = vbTwips,
' we'd want to do txtKeys.SetText = Chr\$(X \ Screen.TwipsPerPixelX)). Don't
' sweat it. Just make sure your callback target's ScaleMode is vbPixels!

Status.SelStart = Len(Status.Text)
Status.SelText = Chr\$(X)

End Sub

Private Sub EnableDialer_Click()

If PickMSIM.Value = False And PickDialpad.Value = False And PickPhoneFree.Value = False Then
MsgBox "You must select a program first"
Exit Sub
End If

PortNumber = Text1.Text
If Form1.MSComm1.PortOpen = True Then Form1.MSComm1.PortOpen = False
Form1.MSComm1.CommPort = PortNumber
If Form1.MSComm1.PortOpen = False Then Form1.MSComm1.PortOpen = True

'start capturing tones

If ttBeginRecognition = ttscFailure Then
MsgBox "Failed to start DTMF recognition! Is your sound system in use by another program?" &
vbNewLine & "Error " & CStr(ttGetErrorVal), vbExclamation
Exit Sub
End If
Status.Text = ""
dialing = True
If PickDialpad.Value = True Then Timer1.Enabled = True

```

If PickMSIM.Value = True Then Timer2.Enabled = True
If PickPhoneFree.Value = True Then Timer3.Enabled = True
EnableDialer.Enabled = False
Command2.Enabled = False
Text1.Enabled = False
Text2.Text = "Listening on com port " & PortNumber

```

End Sub

```

Private Sub Command2_Click()
PortNumber = Text1.Text
Call ringit
Call ringit ' sometimes the detection circuit needs a second ring
End Sub

```

```

Private Sub ringit()
'rings the handset until answered
If MSComm1.PortOpen = True Then MSComm1.PortOpen = False
'Change Comm port here
MSComm1.CommPort = PortNumber
If MSComm1.PortOpen = False Then MSComm1.PortOpen = True
Dim Commhandle As Long
Commhandle = Form1.MSComm1.CommID

```

```

Do
Dim Timeout As Single
Timeout = timeGetTime() + 500
Do
RTSOn = SETRTS
Success = EscapeCommFunction(Commhandle, RTSOn)
Delay (25)
RTSOn = CLRRTS
Success = EscapeCommFunction(Commhandle, RTSOn)
Delay (25)
Loop While (timeGetTime() <= Timeout)
Delay (1000)
'Wait for off hook
Loop While MSComm1.DSRHolding = True
If MSComm1.PortOpen = True Then MSComm1.PortOpen = False

```

End Sub

```

Private Sub Reset_Click()
'reset everything and stop timers
ttEndRecognition
If Form1.MSComm1.PortOpen = True Then Form1.MSComm1.PortOpen = False
Call dialtone_off
Timer1.Enabled = False
Timer2.Enabled = False
Timer3.Enabled = False

```

```
EnableDialer.Enabled = True
Command2.Enabled = True
Delay (1000) 'give the phone a second to settle in case they just hung up
Call Text1_Change
End Sub
```

```
Private Sub Command4_Click()
    Dim htmlpath
    htmlpath = App.Path & "index.html"
    frmBrowser.StartingAddress = htmlpath
    frmBrowser.Show
End Sub
```

```
Private Sub Form_Load()
    'set up state variables
    dialing = False
    dialtoneplaying = False
    talking = False
    Text1.Text = 1
    PortNumber = Text1.Text
    Call Text1_Change
```

```
'set up dtmf detector
Dim IEI As ttInitializeExInfo
With IEI
    .StructSize = LenB(IEI)

    .CallbackMethod = ttcPostMessage

    ' CallbackInfo1 is the target window for window-based callbacks
    .CallbackInfo1 = CallbackTarget.hWnd

    ' CallbackInfo2 is the WM to send for window-based callbacks
    .CallbackInfo2 = WM_LBUTTONUP

    ' Use the defaults...
    .ColRatio = ttAutomatic
    .DeviceOverride = ttAutomatic
    .MinNoise = ttAutomatic
    .RowRatio = ttAutomatic
    .SignalToNoise = ttAutomatic
    .SilenceDuration = ttAutomatic
    .ToneDuration = ttAutomatic
End With

If ttInitializeEx(IEI) = ttscFailure Then
    MsgBox "Failed to initialize ttLib!" & vbNewLine & "Error " & CStr(ttGetErrorVal), vbExclamation
    Unload Me
    Exit Sub
End If

' ttInitializeEx will have filled in all the ttAutomatic values, so
```

```
' we can see what they are now if you uncomment the box below...
MsgBox "Tone duration is " & Format$(IEI.ToneDuration * 12.75, "#.##") & vbNewLine & _
"Silence duration is " & Format$(IEI.SilenceDuration * 12.75, "#.##"), _
vbInformation, "Info"
```

```
End Sub
```

```
Private Sub PickDialpad_Click()
    PickMSIM.Value = False
    PickPhoneFree.Value = False
End Sub
```

```
Private Sub PickMSIM_Click()
    PickDialpad.Value = False
    PickPhoneFree.Value = False
End Sub
```

```
Private Sub PickPhoneFree_Click()
    PickMSIM.Value = False
    PickDialpad.Value = False
End Sub
```

```
Private Sub Text1_Change()
' check for DSR on selected port to sense a device
'note that IR ports also give a positive response, it might not be the phone
```

```
    On Error Resume Next
    If Text1.Text = "" Then Exit Sub
    If Text1.Text > 0 And Text1.Text < 3 Then PortNumber = Text1.Text
    If Form1.MSComm1.PortOpen = True Then Form1.MSComm1.PortOpen = False
    Form1.MSComm1.CommPort = PortNumber
    If Form1.MSComm1.PortOpen = False Then Form1.MSComm1.PortOpen = True
    If Form1.MSComm1.DSRHolding = True Then
        Text2.Text = "Device found on port"
        Command2.Visible = True ' display a test ring button
    Else
        Text2.Text = "Device not ready or not found on com port"
        Command2.Visible = False
    End If
    If Text1.Text > 2 Then Text1.Text = 1
End Sub
```

```
Private Sub Timer1_Timer()
If dialing = True And Form1.MSComm1.DSRHolding = False Then

'check to see if I should start dialtone
If dialtoneplaying = False And Status.Text = "" Then Call dialtone_on

'wait for right number of digits, and kill dialtone on the first one
If Len(Status.Text) > 0 Then
    Call dialtone_off
    If Len(Status.Text) > 9 Then
        ' recognition is ended
```

```

ttEndRecognition
'run the application
AppActivate ("Dialpad.com [")
'send it the right keystrokes to dial
SendKeys "{TAB 13}" & "~" & "{BKSP}" & Status.Text & "~", 0
dialing = False
talking = True
End If

```

```

End If ' if flag = true

```

```

End If

```

```

'reset everything on a hangup
If talking = True And Form1.MSComm1.DSRHolding = True Then
    talking = False
    AppActivate ("Dialpad.com [")
    SendKeys "%{F4}", 1 'alt F4 closes window
    DoEvents
    Delay (300)
    AppActivate ("Dialpad")
    SendKeys "%vr", 1
    Delay 10000
    dialing = True
    Call EnableDialer_Click
End If
If dialing = True And MSComm1.DSRHolding = True And Status.Text > "" Then
    Call dialtone_off
    Status.Text = ""
End If
End Sub

```

```

Private Sub Timer2_Timer()
'This timer routine is for Microsoft Instant Messenger/Net2Phone
'don't waste time if phone is on hook
If dialing = True And Form1.MSComm1.DSRHolding = False Then

'check to see if I should start dialtone
If dialtoneplaying = False And Status.Text = "" Then Call dialtone_on

'wait for right number of digits, and kill dialtone on the first one
If Len(Status.Text) > 0 Then
    Call dialtone_off
    If Len(Status.Text) > 10 Then
        ' recognition is ended
        ttEndRecognition
        'run the application
        AppActivate ("MSN Mess")
        SendKeys "%TCD", 1
        AppActivate ("Phone Call")
        SendKeys Status.Text & "~"
        dialing = False
        talking = True
        Status.Text = ""
    End If

```

```

End If ' if flag = true
End If

```

```

If talking = True And Form1.MSComm1.DSRHolding = True Then
    talking = False
    AppActivate ("Phone Call")
    SendKeys "%U", 1
    Delay (500)
    EnableDialer_Click
End If
    If dialing = True And MSComm1.DSRHolding = True And Status.Text > "" Then
        Call dialtone_off
        Status.Text = ""
    End If
End Sub

```

```

Private Sub Form_Unload(Cancel As Integer)

```

```

'shut down and release audio hardware
    If Form1.MSComm1.PortOpen = True Then Form1.MSComm1.PortOpen = False
    Call dialtone_off
    ' Make sure recognition is ended!
    ttEndRecognition
End Sub

```

```

Private Sub dialtone_on()

```

```

'plays the dialtone wav
    On Error Resume Next
    dialtoneplaying = True
    'Declare sound playing api variables
    SND_ASYNC = &H1
    'Play the sound asynchronously -- return immediately after beginning to play the sound and have it play
    in the background.
    SND_FILENAME = &H20000
    'lpzName is a string identifying the filename of the .wav file to play.
    SND_LOOP = &H8
    'Continue looping the sound until this function is called again ordering the looped playback to stop.
    SND_ASYNC must also be specified.
    SND_PURGE = &H40
    'Stop playback of any waveform sound. lpzName must be an empty string. Dim SYNC As Long
    Dim wav As String
    wav = "c:\windows\dialtone.wav"
    If wav = "" Then Exit Sub
    SYNC = SND_ASYNC Or SND_LOOP
    Dim R As Long
    R = PlaySound(wav, 0, SND_FILENAME Or SND_ASYNC Or SND_LOOP)

```

```

    If R = 0 Then MsgBox ("Error! I couldn't get control of the speaker to play dialtone, but the call may still
work. You may have to stop other audio programs or check device settings in Control panel, Multimedia,
Playback (Use preferred device only)")

```

```

End Sub

```

```

Private Sub dialtone_off()

```

```

    On Error Resume Next
    dialtoneplaying = False
    'Declare sound playing api variables
    SND_ASYNC = &H1
    'Play the sound asynchronously -- return immediately after beginning to play the sound and have it play
    in the background.
    SND_FILENAME = &H20000

```

```

'pszName is a string identifying the filename of the .wav file to play.
SND_LOOP = &H8
'Continue looping the sound until this function is called again ordering the looped playback to stop.
SND_ASYNC must also be specified.
SND_PURGE = &H40
'Stop playback of any waveform sound. pszName must be an empty string. Dim SYNC As Long
SND_NODEFAULT = &H2

```

```

wav = "c:\windows\off.wav"
R = PlaySound("", 0, SND_PURGE Or SND_NODEFAULT)
End Sub

```

```

Private Sub Timer3_Timer()
' this timer is for Phonefree
' don't waste time if phone is on hook

```

```

If dialing = True And Form1.MSComm1.DSRHolding = False Then
'check to see if I should start dialtone

```

```

If dialtoneplaying = False And Status.Text = "" Then Call dialtone_on
'wait for right number of digits, and kill dialtone on the first one

```

```

If Len(Status.Text) > 0 Then
Call dialtone_off

```

```

If Len(Status.Text) > 10 Then
' recognition is ended
ttEndRecognition
'run the application
AppActivate ("PhoneFree")
SendKeys Status.Text & "~"
dialing = False
talking = True
Status.Text = ""
End If

```

```

End If ' if flag = true
End If
If talking = True And Form1.MSComm1.DSRHolding = True Then
talking = False

```

```

' Uncomment this when Phonefree gets their hotkeys working again
'In the meantime, just hope the other party hangs up within 15 seconds

```

```

'AppActivate ("PhoneFree")
'SendKeys "+H", 1
Delay (3000)
'SendKeys "%N", 1
'EnableDialer_Click
Call Reset_Click

```

```

End If
If dialing = True And MSComm1.DSRHolding = True And Status.Text > "" Then
Call dialtone_off
Status.Text = ""
End If
End Sub

```

Figure 7 DTMF detection software parameter tuning

score	Noise Gate	Noise/Signal	Tone Duration	Silence Duration	Pri/Harmonic Ratio
0	0	1	1	1	0
0	450495	1	1	1	0
0	0	1	1	1	25
0	450495	25	1	1	0
0	450495	50	1	1	0
0	1000000	25	1	1	0
0	1000000	50	1	1	0
0	0	1	10	1	12
0	0	1	10	1	25
0	495050	12	5	1	0
0	495050	12	10	1	0
0	495050	12	1	5	0
0	495050	12	1	10	0
0	495050	12	1	1	12
0	495050	12	1	1	25
1	1000000	1	1	1	0
1	0	1	1	1	12
1	450495	1	1	1	25
1	0	1	5	1	25
1	0	1	1	5	25
1	0	1	10	10	0
1	0	1	1	10	12
1	0	1	1	10	25
2	0	1	1	.5	0
2	0	25	10	1	0
2	1000000	1	10	1	0
2	0	1	10	5	0
3	0	25	1	1	0
4	450495	1	10	1	0
4	0	25	1	10	0
4	0	1	5	1	12
5	0	1	1	10	0
5	450495	1	1	1	12
5	0	50	1	10	0
6	0	25	1	5	0
6	0	25	1	1	25
6	0	1	1	5	12
6	0	50	10	1	0
7	1000000	1	1	1	12
7	0	50	1	1	12
7	0	50	1	1	25
8	0	50	1	1	0
8	0	1	5	1	0
8	0	25	1	1	12
8	0	1	5	10	0
8	0	50	5	1	0
9	0	1	10	1	0
9	0	1	5	5	0
9	1000000	1	5	1	0
9	1000000	1	1	5	0

9	1000000	1	1	10	0
9	1000000	1	1	1	25
9	0	50	1	5	0
10	450495	1	5	1	0
10	450495	1	1	5	0
10	450495	1	1	10	0
10	0	25	5	1	0

[illegible]

Figure 8 DTMF Detection Library function

```
// -----
// Main.c - Murphy McCauley
// Sun, Oct 1 / 2000
// The interface and heart of ttLib
// -----

// -----
// Misc
// -----

// This is the main file
#define MAIN

// -----

// -----
// Includes
// -----

#include "ttLib.h"
#include "Main.h"
#include "DFT.h"
#include "Util.h"

#include <math.h>           // For floor() (I think that's all...)
#include <process.h>        // For threading
#include <mmsystem.h>       // For audio

// -----

// -----
// Callback settings (File scope variables)
// -----

// This is a function pointer to the function to call when a key has been
// recognized. This function implements a callback behavior.
GotKeyType      pGotKeyFunc;

// If the callback method is one that calls a function, this is a pointer to
// the function to call.
CallbackType    pCallbackFunc;

// If the callback method is one that notifies via a window, this is the
// window that receives the notification.
HWND           pCallbackWindow;

// If the callback method is one that notifies via a window, this is the
// window message to send.
int             pCallbackMessage;
```

```

// -----

// -----
// Misc variables (File scope variables)
// -----

// WAVEHDRs for the chunks
WAVEHDR    pWaves[NUM_CHUNKS];

// Pointer to the entire audio data buffer (for all chunks)
short * pWaveData;

// Pointers within pWaveData -- one for each chunk
short * pChunkData[NUM_CHUNKS];

// Handle of the audio input device
HWAVEIN phDev;

// The waveform format
WAVEFORMATEX pFormat;

// When TRUE, the decoding loop will exit
volatile BOOL pQuitNow = FALSE;

// Handle of the recognizer thread
HANDLE phThread;

// Audio device to open
UINT pAudioDevice = WAVE_MAPPER;

// Translates a row and column into a key
static char pKeys[4][4] = {{ '1', '2', '3', 'O' }, // O - Flash Override (FO/A)
                           { '4', '5', '6', 'F' }, // F - Flash (F/B)
                           { '7', '8', '9', 'I' }, // I - Immediate (I/C)
                           { '*', '0', '#', 'P' } }; // P - Priority (P/D)

// -----

// -----
// Recognizer settings (File scope variables)
// -----

// This is a noise gate to stop excessively noisy signal from being
// considered as a tone candidate.
int pMinNoise = DEFAULT_MIN_NOISE;

// This is the number of sampling intervals that must pass until
// a key is considered pressed (currently a 12.75ms sampling interval
// is hard coded). I suggest 3 -- 3 * 12.75 = 38.25
int pToneDur = DEFAULT_TONE_DURATION;

// This is the number of sampling intervals that must pass between
// keys (currently a 12.75ms sampling interval is hard coded).
// I suggest 3 -- 3 * 12.75 = 38.25
int pSilenceDur = DEFAULT_SILENCE_DURATION;

```

```
// This determines how much stronger the detected tones must be
// compared with the rest of the signal. Likely range is 10 - 50.
int pSignalToNoise = DEFAULT_SIGNAL_TO_NOISE;
```

```
// These are the ratio between the primary tone and its harmonic.
// The higher they are, the stronger the primary tone must be in
// comparison, which will make the thing less likely to misfire.
// Likely range is 0 - 25. In my testing, most sources are noisy
// enough to throw this off rather easily for at least on frequency.
// I suspect this is due to the tone generators.
int pRowRatio = DEFAULT_ROW_RATIO;
int pColRatio = DEFAULT_COL_RATIO;
```

```
// -----
```

```
// -----
// Misc. internal functions
// -----
```

```
int CalcNumSamples(double MS, int Rate) {
    double T;
    T = MS * (Rate / 1000.0);
    if (floor(T) < T) {
        return (int)(floor(T) + 1);
    } else {
        return (int)floor(T);
    }
}
```

```
unsigned __stdcall ThreadProc (LPVOID Dummy) {
    BOOL TempB;

    TempB = Recognize();

    if ( (TempB == FALSE) && (gLastError == tteNoError) ) {
        SetError(tteUnknown);
    }

    _endthreadex(TempB);
    return TempB; // Just for style
}
```

```
// -----
```

```
// -----
// Callback handling functions
// -----
```

```
// These functions (xCB()) implement callback methods
void SendMessageCB (char Key) {
    SendMessage(pCallbackWindow, pCallbackMessage, Key, Key);
}
void PostMessageCB (char Key) {
    PostMessage(pCallbackWindow, pCallbackMessage, Key, Key);
}
```

```

void DirectFunctionCB (char Key) {
    pCallbackFunc(Key);
}

// Used to set set up a callback method -- a fairly silly (but
// straightforward) implementation.
BOOL SetCallback(ttCallbackMethods Method, int CB) {
    switch (Method) {
        case ttcPostMessage:
            pCallbackWindow = (HWND)CB;
            pGotKeyFunc = PostMessageCB;
            if (!pCallbackWindow) {
                SetLastError(tteBadParameter);
                return FALSE;
            }
            break;
        case ttcSendMessage:
            pCallbackWindow = (HWND)CB;
            pGotKeyFunc = SendMessageCB;
            if (!pCallbackWindow) {
                SetLastError(tteBadParameter);
                return FALSE;
            }
            break;
        case ttcDirectFunction:
            pCallbackFunc = (CallbackType)CB;
            pGotKeyFunc = DirectFunctionCB;
            if (!pCallbackFunc) {
                SetLastError(tteBadParameter);
                return FALSE;
            }
            break;
        default:
            SetLastError(tteNotSupported);
            return FALSE;
    }

    return TRUE;
}

```

// -----

// -----
// Interface functions (prototyped in ttInterface.h)
// -----

```

TouchToneErrors EXPORT ttGetErrorVal() {
    TouchToneErrors ret;
    ret = gLastError;
    gLastError = 0;
    return ret;
}

```

```

BOOL EXPORT ttInitializeEx (ttInitializeExInfo * Info) {
    SetLastError(tteNoError);
}

```

```

    if (Info->StructSize != sizeof(ttInitializeExInfo)) {
        SetError(tteNotSupported);
        return FALSE;
    }
    if (!SetCallback(Info->CallbackMethod, Info->CallbackInfo1)) {
        return FALSE;
    }

    pCallbackMessage= autosetp(&Info->CallbackInfo2, DEFAULT_CALLBACK_MESSAGE);

    pMinNoise          = autosetp(&Info->MinNoise,          DEFAULT_MIN_NOISE);
    pToneDur           = autosetp(&Info->ToneDuration, DEFAULT_TONE_DURATION);
    pSilenceDur        = autosetp(&Info->SilenceDuration, DEFAULT_SILENCE_DURATION);
    pSignalToNoise     = autosetp(&Info->SignalToNoise, DEFAULT_SIGNAL_TO_NOISE);
    pRowRatio          = autosetp(&Info->RowRatio,          DEFAULT_ROW_RATIO);
    pColRatio          = autosetp(&Info->ColRatio,          DEFAULT_COL_RATIO);

    pAudioDevice       = (UINT)autosetp(&Info->DeviceOverride, DEFAULT_DEVICE);

    return TRUE;
}

BOOL EXPORT ttInitialize (ttCallbackMethods Method, int CB) {
    SetError(tteNoError);

    if (!SetCallback(Method, CB)) {
        return FALSE;
    }

    pCallbackMessage = DEFAULT_CALLBACK_MESSAGE;
    pMinNoise        = DEFAULT_MIN_NOISE;
    pToneDur         = DEFAULT_TONE_DURATION;
    pSilenceDur      = DEFAULT_SILENCE_DURATION;
    pSignalToNoise   = DEFAULT_SIGNAL_TO_NOISE;
    pRowRatio        = DEFAULT_ROW_RATIO;
    pColRatio        = DEFAULT_COL_RATIO;

    pAudioDevice     = DEFAULT_DEVICE;

    return TRUE;
}

BOOL EXPORT ttBeginRecognition () {
    int ThreadID;

    if (phThread) {
        SetError(tteWrongState);
        return FALSE;
    }

    pFormat.wFormatTag = WAVE_FORMAT_PCM;
    pFormat.nChannels = 1;
    pFormat.nSamplesPerSec = 11025; // 11kHz
    pFormat.wBitsPerSample = 16;
    pFormat.nBlockAlign = (pFormat.nChannels * pFormat.wBitsPerSample) / 8;
    pFormat.nAvgBytesPerSec = pFormat.nBlockAlign * pFormat.nSamplesPerSec;

```

```

pFormat.cbSize = 0;

    if (!phDev) {
        waveInOpen(&phDev, pAudioDevice, &pFormat, 0, 0, CALLBACK_NULL);
    }
    if (!phDev) {
        SetError(tteWaveDeviceDidNotOpen);
        return FALSE;
    }

    waveInStart(phDev);

    phThread = (HANDLE)_beginthreadex(
        NULL,
        0,
        ThreadProc,
        NULL,
        0,
        &ThreadID);

    if (phThread) {
        return TRUE;
    } else {
        waveInReset(phDev);
        waveInClose(phDev);
        phDev = 0;
        return FALSE;
    }
}

BOOL EXPORT ttEndRecognition () {
    if (!phThread) {
        // It's not running!
        SetError(tteWrongState);
        return FALSE;
    }

    // Tell the thread to quit
    pQuitNow = TRUE;

    // Wait for the thread to finish, then clean it up
    WaitForSingleObject(phThread, INFINITE);
    CloseHandle(phThread);
    phThread = NULL;

    // Reset this...
    pQuitNow = FALSE;

    // Close and clean up the wave device
    if (phDev) {
        waveInClose(phDev);
        phDev = NULL;
    }

    return TRUE;
}

// -----

```